14 COMMON VERSIONING OF PRODUCT DATA AND ENGINEERING PROCESSES

Martin Eigner[a] and Fabrice Mogo Nem[b]

University of Technology Kaiserslautern, Department of Mechanical and Process Engineering, Institute for Virtual Product Engineering, Gottlieb-Daimler-Str/Geb.44, 67663 Kaiserslautern, Germany, P.O. Box 3049, 67653 Kaiserslautern, Germany, Tel: +49-(0)631-205-[1]3873/[2]2312, Fax: +49-(0)631-205-3872.
E-mail: [a]eigner@mv.uni-kl.de, [b]mogo_nem@mv.uni-kl.de

PDM/PLM (Product Data/Lifecycle Management) and Workflow Management (WM) systems have been established both as backbone for the management of product data and their related engineering processes. Both in collaborative and multidisciplinary engineering contexts, changes occurring on product data and engineering processes must be tracked to fulfill (inter)national legislations such as (re)configuration management. Versioning is a key solution to achieve it. However its separate implementation in both systems led to less traceability in engineering. Although the advantages of integrating PDM/PLM with WM systems have been noticed and addressed, the way changes have to be tracked and versions be managed in such an integrated context has been neglected so far. A comprehensive understanding of existing versioning concepts in product data and engineering process management is necessary for addressing it. This paper reviews some of these concepts and proposes a semantic versioning concept for our integrated product and process metamodel called Engineering Networks.

Keywords: Product Data Management, Workflow Management, Versioning, Engineering Networks.

1. INTRODUCTION

Today's globalized world and resulting perturbations in the business environment drive enterprises to steadily reduce the time and costs for product development as well as to be flexible in the deployment of their business processes. PDM/PLM (Product Data/Lifecycle Management) and WM (Workflow Management) systems have therefore been established in enterprises respectively for the management of the product relevant information throughout the product lifecycle and for the modeling and deployment of the engineering processes they walk through. The increasing complexity of products in collaborative and multidisciplinary product engineering contexts (e.g. Mechatronics) and the rising demands for more product reliability and product liability raised new challenges. In case for example of the reconfiguration of a product configuration after a crash according to product liability laws, not only the states of the relevant items at the time the product was built could be of interest but their historical evolvement too (e.g. the nature and the version of processes they walked through reaching these states). Gathering such information from nowadays separated involved and distributed WM and PDM/PLM systems can be a real tricky and complex task. A tighter management of product data with their engineering processes is therefore required. This has been noticed and the usage of a common metamodel for defining both the product data model and their engineering process models and thus for enabling the integration of PDM/PLM with WM systems in product engineering has been proposed.[1] However, existing approaches in this area have mostly focused on modeling aspects whereas semantic concepts describing the way changes on product data, product data model as well as changes on associated engineering process models must be managed and propagated have been neglected. This paper reviews some concepts and terminologies of versioning in product data and engineering process management and proposes a semantic concept for their coequal implementation in an integrated product and process environment.

The remainder of this paper is organized as follows: in Section 2 we introduce the concept of versioning in general. Then, we separately review versioning in product data and engineering process management and analyze their similarities. In Section 3, we firstly introduce our integrated modeling concept for product data and engineering processes called Engineering Networks, and secondly propose a semantic concept for versioning in it. We finish this paper we a conclusion and outlook in Section 4.

2. VERSIONING

Versioning is likely one of the most important concepts in the field of information management. It is needed in application areas where not only the current state of a data (called a *design object*) is important but its previous states (called *versions*) as well.[1] Katz defines the terms design object and version as follows: "*A design object is an aggregation of design data treated as a coherent unit by designers. Over time, its semantically meaningful snapshots form versions*."[2] In this paper, we simply use the term *object* instead of *design object* and assume it to be an instance of a given object type (a schema or a class in terms of object orientation). An object under version control is called a *versioned object*. A version denotes a specific state of a versioned object and can be seen itself as a versioned object for its successor versions.

Many disciplines are nowadays concerned with versioning. Thus, an object can represent for example a workflow, a Computer-Aided-Design (CAD) document, a source or an executable file, a Very-High-Speed-Integrated-Circuit-Hardware-Description-Language (VHDL) design document, or even a hierarchical structure of mechanical, electrical and software items in case of Mechatronics. There are various reasons for creating a new version of a versioned object. In most of the cases a new version results from changes to the previous one. Instead of overwriting the old states of that versioned object, they are preserved and managed for different purposes. Depending on the application field, only one (mostly the newest) or many versions of a versioned object can be in use at a given time. Conradi uses the term *Revision* for a version intended to supersede its predecessor and the term *Variant* for a version which is intended to coexist with its predecessor.[5] Due to the non consistent usage of the terms revision and variant in the literature, we don't make a difference between a version and a revision in this paper. A *Version model* defines the versioned objects, version identification and organization, as well as operations for retrieving existing versions and constructing new versions. It consists of a *product* and a *version space*.[5] The product space represents the objects to be versioned and their relationships. The version space contains the versions of versioned objects and determines the way they are organized. Usually, versions of an object are organized in an acyclic graph called a *version graph*. It consists of nodes and edges and records the history of changes operated on that object throughout its lifecycle. The nodes in the graph represent the versions and the edges the dependencies (e.g. ancestor-successor) between them. Due to time and space optimization reasons, a version model can be state- or change-based. In the former the states of each object version are stored. In the latter only the differences (called the *delta*) between predecessor and successor versions are stored. The remainder of this section analyzes versioning in product data and engineering process management.

2.1. Versioning in Product Data Management

Product data management (PDM) is the discipline responsible for the management of product data in engineering disciplines such as Mechanics (ME) or Electrics/Electronics (E/E). For a long time it has been restricted to data related to design and manufacturing. In order to support all the activities during the whole product lifecycle from the requirement phase up to recycling, the scope of PDM has been extended to Product Lifecycle Management (PLM). With the introduction of mechatronics, product lifecycle management should support the three disciplines ME, E/E and Software Engineering (SE). Although the discipline of SE was kept apart and evolved separately, similarities between PDM/PLM and software configuration management (SCM) exist.[8,9] Both disciplines deal with complex data structures and are concerned with controlling access and changes on data. Although SCM is mostly concerned with versioning textual files instead of complex object structures as it is the case in PDM/PLM, both can be assumed to deal with objects having hierarchical and/or networked structures of (inter)related objects. Due to the heterogeneity of the three engineering disciplines involved in mechatronics, it remains a big challenge to develop a global product model mapping the mechatronical product data. We consider however for our analysis the product space in PDM/PLM as a hierarchical structure of (inter)related (vertically and/or horizontally), distributed and multidisciplinary objects.

Analyzing versioning in PDM/PLM requires distinguishing between the modeling abstraction level concerned, the granularity of versioned objects and the concepts used for version identification and change propagation. In PDM/PLM versioning is used in general to manage the changes occurring during the management of product lifecycles. Two of the following four modeling abstraction levels *Meta-metamodel*, *Metamodel*, *Model* and *Instance* as defined in Ref. 15 are therefore directly concerned. Versioning at Instance level can be seen as the backbone for the realization of change and configuration management as described by standards and recommendations like for example DIN 672, DIN ISO 9001, EG-Norm 85/374, EN ISO 10007 and SASIG ECM Recommendation. Changes at this level as described in DIN 6772 can be initiated for failure adjustment, customer or market needs, or after changes in legislatures.[14] Example giving, after some problems have been reported by customers, the version (VerA) of car current being manufactured and offered to the customers is changed and a new version (VerB) reflecting the changes is released for future manufacturing processes. At the next higher abstraction level, versioning deals with the evolution of the product data models (schemas) use to describe the product data at *Instance* level. Example giving, in order to support a better management of the overall costs of a car throughout its lifecycle, a company extends its underlying product data model for car and adds an additional attribute for capturing the recycling costs to the already existing set of attributes. Further, considering the today's contexts of configurable products enabling customers to individually configure their product according to a specific product family definition, this strong separation of the two abstraction levels Instance and Model in PDM/PLM becomes more complex as in conventional data modeling perception. A product instance is instantiated in such a case according to a product family description (also called configuration model) which in turn can be considered as being an instance of a specific product data model.[3] For complexity reason, only versioning in product data management at Instance level is addressed in this paper. The evolvement of product data models and the consideration of configurable products will be addressed in future works.

Considering our assumption of hierarchical structure of (inter)related objects building a whole product, the granularity of versioned objects influences the required versioning concept. We distinguish between versioning an atomic object, a composite object or even a relationship between objects. The former addresses the versioning of objects with atomic structure i.e. without other versioned objects as component. It can be seen as the simplest case due to the non existence of dependencies between objects in contrast to versioning a composite object. A composite object consists of other versioned objects and its state at a given time is called a *configuration*. Change on a component in a composite object can initiate cascading changes on other objects. Although versioning a composite object is also concerned with versioning atomic objects, it requires mechanisms to deal with propagation of changes which are addressed later. Versioning a composite object can however be seen as a special case of versioning a relationship between objects. It represents the versioning of the vertical *part-of* relationship between an object and its component objects. The consideration of the overall product lifecycle and the multidisciplinarity require paying more attention to this granularity of versioning in the future. Depending on the nature of the relationship, change on an object can affect in a specific way the objects (inter)related to it. Realizing this requires the modeling of the semantic of relationships which can be achieve using powerful modeling constructs or ontology.

Realizing versioning requires a mechanism for the unique identification of a versioned object and their versions. Further such a mechanism should enable the decision whether two versions belong to the same versioned object. This is realized usually by assigning unique object identifier (OID) to versioned objects and their versions. In order to be able to decide whether or not two object versions belong to the same versioned object the OID is often subdivided in a version independent part and a version specific part. The version independent part (ID) is generated by the versioning system and assigned to each new created versioned object and remains the same during its whole lifecycle (e.g. a new part with part identifier 4711). For each its versions a unique version identifier (VID) is generated and together with the identifier ID is used to uniquely identify a version (e.g. the first third versions of the part with part identifier 4711 are 4711.**1**, 4711.**2** and 4711.**3**). The management of the two identifiers is not as easy as described above. It embodies the underlying mechanism dealing with version evolvement and propagation of changes. In the case of versioning an atomic object such a mechanism can be applied. In case of versioning a composite object or a relationship in general, a change on one object can initiate changes on other (inter)related objects. In order to limit the impact of such a change to only few objects the notion of compatibility between object versions is used.[14] The problem of binding an object configuration to its versioned components is also interesting in versioning. It can be *static* i.e. to a specific version of the component, *dynamic* i.e. to the newest version of the component or in general to settings that can be resolved at runtime, *event triggered* i.e. after the occurrence of a specific event, or *time triggered* i.e. at a given time.

2.2. Versioning in Engineering Process Management

The Workflow Management Coalition[6] defines the term business process as "*A set of one or more linked procedures or activities which collectively realize a business objective or policy goal ...*". Its computerized facilitation or automation of a business process, in whole or in part is called *Workflow*.[7] An instance of a Workflow is called a *Case* and corresponds to the execution of some works according to the underlying workflow definition called *Workflow Model* or *Process Model*. We use the term process for both a process and its associated computerized automation workflow and restrict our analysis to processes in engineering called engineering processes.

The modeling of a process is itself a process. That means a process is rarely modeled perfectly at once and undergoes different changes during its lifecycle. Due to the fast changes in today's globalized world, enterprises are driven to be able to react quickly by adapting their processes in order to stay competitive. Thus, managing the evolution of process models and their impact on process instances has become a necessity.[8] In nearly every process modeling context, a process is considered as a composition of tasks, roles and links (routing or data flows). Some modeling languages allow the definition of process hierarchy, i.e., a process can recursively consist of another (sub) processes. We consider a process as a set of tasks, roles, links and (sub) processes, and a process model change as any modification (insertion, deletion or reordering of tasks, etc.) of an existing process model. We distinguish three categories of process changes: *evolutionary changes*, *temporary changes* and *changes aiming at creating process variants*. For a better understanding let us consider the example of a production process in an enterprise manufacturing mechatronical car's engine. The enterprise owns at beginning only a single process which evolves over time (*see Figure 1*).

The reasons behind an evolutionary changes are numerous: changing business context (e.g. demand of individual customers), changing legal context (e.g. new legislature) or changing technological context (e.g. change in technical infrastructures).[9] Changes aiming at resolving modeling errors detected during process deployment can also be assigned to this category. In this case the underlying process model must be permanently changed and all the cases started subsequently follow the new process model. In the research and praxis, expiration and effective dates are used to deal with this category of change. In Figure 1 part (a), the initial production process (VerA) is permanently changed and a new version (VerB) is created. VerB will become effective at t_{Eff}. That means at t_{Eff} the process VerA will expire and once all of the running cases tied to it will complete, it will become inactive and can be therefore archived. This first category of changes is supported by many commercial workflow management systems like Oracle Bea and ActiveBPEL. A similar concept is also used in the ISO Standard 10303 (STEP) Part 214 Unit of Functionality S8 for versioning production processes. The second category is concerned with changes aiming at reacting to temporary changes in the process environment. This is typically the case if an enterprise must react quickly to unexpected or unusual events. It addresses the problem of flexibility during process deployment. Figure 1 part (b) illustrates it. Due to, for example, maintenance activities the resources required to execute the task T2 in the process version VerB are

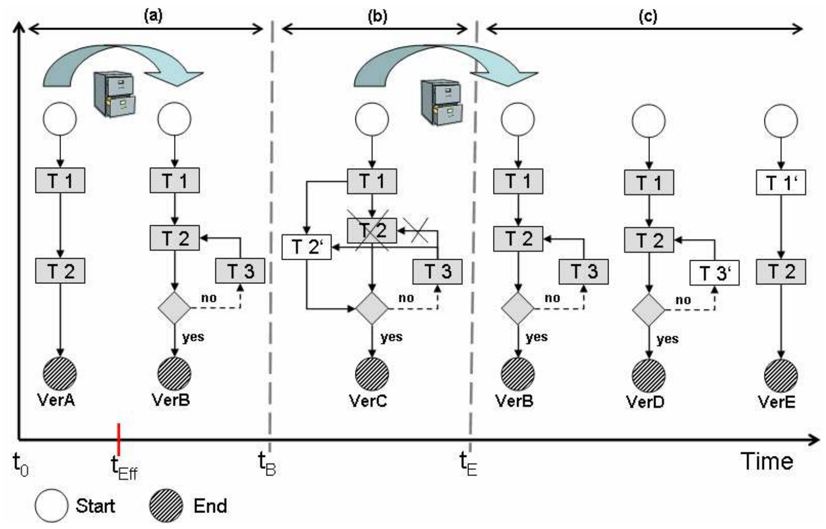

Figure 1. Changes on engineering processes during their lifecycle.

not available. The manager in charge for production decides to skip the execution of T2 and switches to a temporary defined task T2’ for compensation. Such a change affects one or a group of cases and doesn’t lead automatically to a permanent change of the underlying process model (VerB). Only the cases started between t_B and t_E use the new process model VerC. Cases started after t_E use the previous process model VerB. If at least one case has been successfully executed following the process model VerC it must be archived for traceability purposes. The last category of change is concerned with the creation of new variants of a process model. Afterwards, many versions of that process model are valid at the same time (*see Figure 1 part (c)*). For every new case one of the process variants VerB, VerD or VerE can be chosen. This last category can also be seen as extension of the second whereby either after a temporary change the resulting process model is required to be permanently available or during a temporary change new cases must be started according to an old version.

During our analysis some important points have been intentionally omitted so far. The first point concerns the question: What happens with cases running before and after changes are initiated and performed? The second question deals with the problem of identifying the versions. The former has been addressed in many research activities and one of the following solutions can apply[9]: *Forward recovery* (the old cases are aborted and appropriate measures are taken; *Backward recovery* (the old cases are aborted and rolled back or compensated, and then they are restarted according to the new process model); *Proceed* (the old cases are handled the old way, new cases are handled the new way); *Transfer* (the old cases are transferred to the new process model); *Detour* (for momentary changes it is often wise to allow a temporary detour such that the unexpected situation can be cleansed). The second question is of special interest for our analysis. Assuming a hierarchical structure of (flexible) process models where a (sub) process model or even a single task can be used in several process models, only few research activities currently address the problem of numbering the process versions for their identification. For example, instead of using the numbering system consisting of two digits separated by a dot where the first represents the main version and the second the sub version, Zhao proposes a numbering concept consisting of three digits x.y.z.[12] He assumes thereby a process in contrast may evolve along two axes, i.e., the permanent improvement and the temporary adaptation. The first digit x denotes the major version; the second digit y denotes the minor version and the third digit z denotes the temporary variation of a process model. Applying this reasoning to the example in Figure 1 leads to the assignment of the number **1.0** to VerA, **1.1** to VerB, **1.1.1** to VerC, and respectively **1.2** and **1.3** to VerD and VerE. This numbering concept is indeed interesting but it doesn’t take in consideration the situation where a temporary change is applied on a process version resulting from another temporary change. Following this numbering concept leads to the extension of the three digits x.y.z to four digits w.x.y.z and so forth.

2.3. Identified Similarities and Differences in Both Disciplines

After versioning has been analyzed in both disciplines some similarities and differences can be observed according to the criterions: reasons for versioning, modeling abstraction levels concerned with versioning and the granularity of versioning. In both disciplines versioning is used as key concept for keeping track of changes. In PDM/PLM versioning can occur at *Model* level as well as at the *Instance* level as discussed in Section 2.1, whereby we only focused in this paper on versioning at *Instance* level. In engineering process management instead, versioning only occurs at *Model* level. Even though changes on running process cases are allowed at *Instance* level (in context of flexible processes), they directly affect the underlying process model at *Model* level. The last criterion addresses the granularity of versioned objects. In both disciplines we can distinguish between versioning a single item, a composite item or a relationship.

3. VERSIONING IN ENGINEERING NETWORKS

Engineering Networks is a concept currently under development. It aims the increasing of traceability and quality in product lifecycle management and further at reducing the costs for PDM/PLM implementation in Small and Medium sized Enterprises (SME).[16] It supports the coequal modeling of product data and their related engineering process through the definition of a common metamodel. It is based on the two concepts of *Engineering Object (EO)* and *Engineering Process (EP)*. Considering the two modelling abstraction levels *Model* and *Instance*, an EO represents at *Instance* level an object in a given state in its lifecycle (e.g. a mechanical part with number 4711 in its release state and with current version number v5). It is defined at *Model* level by an EO-Type. An EP represents at *Instance* level an instance of an engineering process in a given state (e.g. in execution, terminated or aborted). It is defined at *Model* level by an EP-Type which is related with the corresponding EO-Type definition. Engineering Networks is located at the *Metamodel* level (*see Figure 2*). EO-Type and EP-Type can be considered as an extended type of class in terms of object orientation. In Figure 2 for example five different EO-Types are used at *Model* level to represent a product structure consisting of an EO-Type for a product (PR), an EO-Type for a product function (F) and an abstract EO-Type for physical items in a product (I). The abstract EO-Type I is the parent type in the EO-Type hierarchy consisting of EO-Types for real physical part (P) and assembly of parts (A). The link between the EO-type Types F and I denotes that between a function and a physical item (part or assembly) a dependency may exist. At the same modeling abstraction level, four different EP-Types are used to define a process structure representing an engineering change management process (EP-Type CMP) for a product. It consists of an impact analysis process (EP-Type IAP), a cost analysis process (EP-Type CAP) and an engineering release process (EP-Type ERP). The link between the PR and CMP relates a product with its change process. An EP-Type can also be associated to a relationship (e.g. ERP is associated to the link between A and I). The meaning of such a relationship is illustrated at Instance level in Figure 2. PR1 is an instance of PR and consists of the instances F1, F2, F3 of F. A1, A2 and A3 are instances of A. The instance A2 of A has been released using an instance of ERP and is part of the assemblies A1 and A3. The link between A2 and A3 has additionally been released using an instance ERP for electro magnetic compatibility reasons. Instance of different versions of the EP-Type ERP have been used and such information are crucial for traceability and reconfiguration.

In order to version instances of EO-Types (i.e. EOs) in Engineering Networks as described above, we use the concepts of versioning composite objects and relationships as described in Section 2.1. Identifiers for EOs are therefore composed of a version dependent and a version independent part. In case of compatibility between a new version of an EO and its predecessor, the new emerging EO version shares the same identifier with its predecessor and only a new version identifier is generated. Otherwise

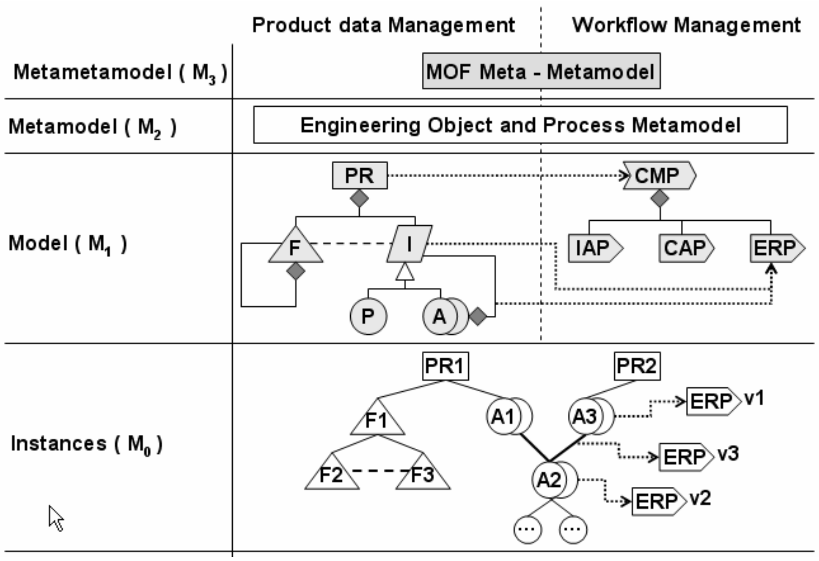

Figure 2. Position of Engineering Networks in the four modeling abstraction levels.

a fully new identifier is created and the emerging EO version is handled further on as a fully new EO. We also apply this concept of versioning composite objects for versioning EP-Types. We need however an analogical concept of compatibility between two versions of a process model. Notions of equivalence between Workflow models have been addressed in details in Ref. 18 The concepts described therein have been kept generic since the aim was the providing of means for assessing equivalence between workflows described using different languages and even different modeling skills and therefore for enabling inter-language mapping of workflows. Focusing on our purpose of defining a concept for assessing compatibility between engineering processes, we define two versions of an engineering process model compatible if they achieve under the same environmental conditions the same goals. We mean with goals the same observable behaviors which can be formally described using workflow traces. This definition is based on a relaxed concept of *Observational Equivalence* as described in Ref. 18 The final assessment of compatibility is leaved to the appreciation of the engineering process administrator. We use this concept in EN for applying the same concept of versioning EO instances to versioning EP-Types and hence to find a common basis for commonly defining them. In Engineering Networks, two types of relationships between product and engineering processes can be distinguished. Those are: the relationship between an EO and an EP at *Instance* level and the relationships between an EO-Type and an EP-Type at Model level. The former is used for traceability purposes in engineering. The latter is defined during product data modeling and relates EO-Types with their associated engineering processes. In Engineering Networks, changing an EO doesn't require changing its underlying EO-Type definition or associated EP-Types. However, changing an EP-Type can affect related EO-Types since some EOs instantiate afterwards could be required to use the new version of the changed EP-Type. This case has been discussed in this paper and relies on the concept of compatibility between two process versions. If after a change a version of an EP-Type is incompatible with its predecessor version, the associated EO-Types are affected by the change too and consequently need to be versioned.

4. CONCLUSIONS AND OUTLOOK

In order to increase quality and traceability in product engineering, a tighter management concept for product data and related engineering processes is required. The usage of a common metamodel for defining both the product data model and their engineering process models enables the integration of PDM/PLM with WM systems and offers good opportunities for achieving it. In order to address the versioning and the propagation of changes in such an integrated context, a review of versioning concepts and terminologies in both product data and process management has been done in this paper. Its result has been used as fundament for proposing a semantic concept dealing with propagation of changes occurring on product data and engineering process models. For complexity reason, only the versioning in product data management at Instance level has been considered. The evolvement of product data models after a change and the consideration of configurable products will be addressed in future works.

REFERENCES

[1] Kovse, J. (2005). Model-Driven Development of Versioning Systems, Ph.D. Thesis, Department of Computer Sciences, Kaiserslautern University of Technology.

[2] Katz, R. H. (1990). Toward a Unified Framework for Version Modeling in Engineering Databases, ACM Computing Surveys, **22**(4), December.

[3] Kovacs, Z., McClatchey, R., Le Goff, J.-M. and Baker, N. (1999). Patterns for integrating manufacturing product and process models, third international conference on Enterprise Distributed Object Computing, proceedings, pp. 37–48.

[4] Männistö, T. and Sulonen, R. (2007). Evolution of Schema and Individuals of Configurable Products, Lecture Notes in Computer Science, 1727:12–23, July 2007.

[5] Conradi, R. and Westfechtel, B. (1998). Version Models for Software Configuration Management, ACM Computing Surveys, **30**(2), June.

[6] WFMC (1999). Workflow Management Coalition — Terminology & Glossary, Document Number WFMC-TC-1011, Document Status — Issue 3.0.

[7] Hollingsworth, D. (1995). Workflow Management Coalition — The Workflow Reference Model, The Workflow Management Coalition Specification, Document Number TC00-1003, Document Status — Issue 1.1, January.

[8] Bea, H. and Bae, J. (2007). A Version Management of business Process Models in BPMS, International Workshops (APWeb/WAIM 2007), Lecture Notes in Computer Science 4537:534–539.

[9] Van der Aalst, W. M. P. and Jablonski, S. (2000). Dealing with workflow change: identification of issues and solutions, International Journal of Computer Systems Science & Engineering, 15(5):267–276, September.

[10] Westfechtel, B. and Conradi, R. (1998). Software Configuration Management and Engineering Data Management: Differences and Similarities, ECOOP'98 SCM-8 Symposium Brussels, Proceedings, Lecture Notes in Computer Science (LNCS), 1439:95–106.

[11] Estublier, J., Favre, J. M. and Morat, P. (1998). Toward SCM/PDM integration?, ECOOP'98 SCM-8 Symposium Brussels, Proceedings, Lecture Notes in Computer Science (LNCS), 1439:95–106.

[12] Xiaohui, Z. and Chengfei, L. (2007). Version Management in the Business Process Change Context, 5th International Conference on Business Process Management (BPM 2007), Proceedings, Lecture Notes in Computer Science (LNCS), 4714:198–213.

[13] Hein,l P., Horn, S., Jablonski, S., Neeb J., Stein K. and Teschke M. (1999). A comprehensive approach to flexibility in workflow management systems, International Joint Conference on Work Activities Coordination and Collaboration (WACC'99), Proceedings, pp. 79–88.

[14] Eigner, M. and Stelzer, R. (2008). Produktdatenmanagement-Systeme: Ein Leitfaden für Product Development und Life Cycle Management, Springer Verlag Berlin/Heidelber.

[15] Object Management Group (OMG) (2002). Meta Object Facility (MOF) Specification, version 1.4.

[16] Mogo, F., Weidlich, R. and Eigner M. (2008). Engineering Networks: A Concept for the Coequal Modeling of Data and Processes in Product engineering, 10th International Design Conference (DESIGN 2008), Proceedings, 2:849–856, Dubrovnik/Croatia.

[17] Hidders, J., Dumas, M., van der Aalst, W. M., Hofstede, A. H. and Verelst, J. (2005). When are two workflows the same, 11th Australasian symposium on Theory of computing, Proceedings, ACM International Conference Proceeding Series, 105:3–11, Newcastle.